Лабораторная работа № 2

Программирование с использованием классов.

**Цель работы:** Получить практические навыки работы с классами.

## Теоретические сведения

В языке С++ класс стал расширением понятия структуры, он задает некоторый структурный тип данных (**данные-члены класса**) и набор операций над ними (**функции-члены класса или методы**).

Тип данных класс можно определить с помощью конструкции

**ключ\_класса имя\_класса{ список \_членов };**

Здесь **ключ\_класса** - одно из служебных слов **struct, union, class;**

**имя\_класса** - произвольный идентификатор;

**список\_членов** - определения и описания членов класса, как данных так и функций.

Класс - это набор из одной или более переменных и функций, возможно, различных типов, сгруппированных под одним именем.

Пример структуры - учётная карточка сотрудника, в которой содержится Ф.И.О., адрес, должность, год поступления на работу и т.д. Некоторые из этих атрибутов сами могут оказаться структурами. Так, Ф.И.О. имеет три компоненты, адрес - также несколько компонент.

Каждый из данных-членов класса распознаётся по своему имени, которое должно быть уникальным в данном классе. Члены класса часто называются его элементами или полями.

Хотя каждому имени поля класса сопоставлен определённый тип, самостоятельным объектом такой элемент не является. Память выделяется только для конкретного объекта класса в целом.

Введем новые типы **FIO** и **sotrudnik**:

struct FIO

{ char \*familia, // указатели на строки с фамилией,

\*imya, // именем и отчеством

\*otchestvo;

}; //-------------------- endFIO

struct sotrudnik

{ FIO name; // ФИО сотрудника ( три указателя )

char \*dol; // должность ( указатель на строку)

float okl; // оклад (сумма оклада)

};//---------------- end sotrudnic

Заметим, что наличие "**;**" после фигурных скобок здесь обязательно.

Теперь можно объявить структурные переменные типа **FIO** или **sotrudnik**, то есть выделить память для них, также как это делается для обычных переменных:

int \_tmain(int argc, \_TCHAR\* argv[])

{

FIO chel;

sotrudnik s;

int nf,ns;

cout<<sizeof (FIO)<<'\t'<<sizeof (s);

system ("pause");

return 0;

}

Компилятором будет выделена память под переменные **chel** (тип **FIO**), **s**(тип **sotrudnik**). Отметим, что число байтов, выделяемое под структурную переменную, не всегда равно сумме длин отдельных элементов структуры из-за эффекта выравнивания, производимого компилятором. Чтобы определить реально выделенное число байтов, надо воспользоваться операцией sizeof, например, так:

int nf=sizeof (FIO), ns=sizeof (s);

Для заполнения полей, добавим в класс FIO метод ввода данных с клавиатуры:

struct FIO

{. . .

void define\_fio()

{ char str[40];

cout<<endl;

cout<<"Фамилия :";

cin>>str;

familia=new char[strlen(str)+1];

strcpy (familia,str);

cout<<"Имя :";

cin>>str;

imya=new char[strlen(str)+1];

strcpy (imya,str);

cout<<"Отчество :";

cin>>str;

otchestvo=new char[strlen(str)+1];

strcpy (otchestvo,str);

}

}; //-------------------- endFIO

А в класс sotrudnik два метода :

* define() будет заполнять данные-члены класса
* display() будет отображать их на экране.

struct sotrudnik

{. . .

void define(FIO& fnam, char \*pd, float ok)

{ name.familia = fnam.familia;

name.imya = fnam.imya;

name.otchestvo = fnam.otchestvo;

dol = pd;

okl = ok;

}

void display()

{ cout<<endl;

cout<< name.familia<<" ";

cout<<name.imya;

cout<<" "<<name.otchestvo<<" , ";

cout<<dol<<" , ";

cout<<"оклад :"<<okl<<endl;

}

};//---------------- end sotrudnic

int \_tmain(int argc, \_TCHAR\* argv[])

{

FIO chel; // переменная для ввода Ф.И.О.

sotrudnik s1;

chel.define \_fio();

s1.define(chel,"бухгалтер",10000);

s1.display();

system ("pause");

return 0;

}

Если класс задан ключевым словом struct, то доступ ко всем элементам открыт, то есть к полям и методам класса можно обращаться из функции main():

cout<<chel.familia<<'\t'<<s1. dol<<'\t'<<s1.okl<<'\t'<<s1.name.imya<<endl;

Имена наподобие chel.familia, с помощью которых можно получить доступ к элементам класса, называют уточненными именами.

Рассмотрим работу с использованием указателей, которую можно разделить на три этапа :

* + - * создание указателя : FIO \*fio\_p;
      * установка указателя на объект :fio\_p = &chel;
      * работа с объектом : обратиться к отдельному элементу структуры chel можно двумя способами:

(\*fio\_p). define\_fio();

fio\_p ->familia;

Первый и второй этапы можно объединить :FIO \*fio\_p = &chel;

**Конструкторы и деструкторы.**

Конструкторы предназначены для инициализации абстрактных типов данных, то есть объектов классов.

Деструкторы нужны для удаления локальных объектов в процессе работы, для освобождения динамической памяти, используемой объектом класса.

Имя конструктора должно совпадать с именем класса, конструктор явно или неявно вызывается при определении объекта класса, основное назначение конструктора – выделение участка памяти, превращение его в объект класса, с возможной инициализацией (заданием начальных значений).

В предыдущем примере при создании объектов класса работали автоматические конструкторы, создаваемые компилятором. Для нашего класса sotrudnik автоматический конструктор некорректно создаёт объект s1, так как кроме выделения памяти под указатели: familia,imya,otchestvo требуется выделить память под динамические строки. Почему же не возникала ошибка во время выполнения программы? Давайте разберемся, в примере последовательно вызываются методы:

chel.define\_fio();

s1.define(chel,"бухгалтер",10000);

s1.display();

Вызов **chel.define\_fio();** выделяет память и заполняет динамические строки объекта **chel**. Вызов **s1.define(chel,"бухгалтер",10000);** связывает указатели

**familia, imya, otchestvo**объекта**chel** с указателями **name.familia, name.imya, name.otchestvo**объекта**s1**, таким образом объект **s1**не имеет своей динамической памяти под строки и использует память объекта **chel**

Объект **chel** хотя и имеет свою динамическую память, но получает её не в момент создания объекта, а при вызове метода **define\_fio**(), таким образом, если убрать вызов **chel.define\_fio()**, то программа перестанет работать

Чтобы избежать подобных ошибок необходимо помнить простое правило: **при использовании динамической памяти, её должен выделять конструктор при создании объекта класса.**

struct FIO

{. . .

FIO ()

{cout<<"FIO::Конструктор по-умолчанию"<<endl;

familia=new char[60];

strcpy (familia," ");

imya=new char[60];

strcpy (imya," ");

otchestvo=new char[60];

strcpy (otchestvo," ");

}

~FIO() // деструктор класса FIO

{cout<<endl<<"FIO::Destructor!"<<endl;

delete[] familia;

delete[] imya;

delete[] otchestvo;

}

};//----------------- end FIO

struct sotrudnik

{. . .

// конструктор по-умолчанию

sotrudnik()

{name.familia=new char[60]; // выделение динамической памяти под строку

strcpy (name.familia," "); //инициализация

name.imya=new char[60];

strcpy (name.imya," ");

name.otchestvo=new char[60];

strcpy (name.otchestvo," ");

dol=new char[60];

strcpy (dol," ");

okl = 0;

}

~sotrudnik() // деструктор класса sotrudnik

{cout<<endl<<"sotrudnik::Destructor!"<<endl;

delete[] name.familia;

delete[] name.imya;

delete[] name.otchestvo;

}

};//---------------- end sotrudnic

int \_tmain(int argc, \_TCHAR\* argv[])

{ FIO chel();

chel.define\_fio();

sotrudnik s1;

s1.define(chel,"бухгалтер",12000);

s1.display();

system ("pause");

return 0;

}

После запуска программы, увидим, что конструктор типа FIO вызывается дважды: при вызове **FIO chel() и sotrudnik s1** (работают конструкторы по-умолчанию)

**Работа с массивами абстрактных типов.**

Рассмотрим вариант программы, которая работает с массивом объектов типа **sotrudnik**. Массивам абстрактного типа нельзя присвоить начальные значения при объявлении, они инициализируются в два этапа:

* выделяется память с помощью конструктора по-умолчанию;
* задаются начальные значения с помощью метода класса, в нашем случае с помощью define().

Итак, потребуется конструктор по-умолчанию, то есть конструктор без параметров.

int \_tmain(int argc, \_TCHAR\* argv[])

{FIO chel();

const int n=3;

sotrudnik M[n];

char M\_dol[3][15]={"бухгалтер","слесарь","курьер"};

int M\_okl[3]={60000,15000,10000};

for (int i=0;i<n;i++)

{cout<<"сотрудник\_"<<i+1<<" :";

chel.define\_fio();

M[i].define(chel, M\_dol[i], M\_okl[i]);

}

for (int i=0;i<n;i++) M[i].display();

system ("pause");

return 0;

}

**Доступ к полям класса**

В С++ можно ограничить видимость данных и функций класса при помощи спецификаторов доступа **public, protected, private**. Спецификатор доступа применяется ко всем элементам класса, следующим за ним, пока не встретится другой спецификатор, либо пока не кончится определение класса.

Итак, для изменения видимости полей данных можно явно использовать ключи доступа:

**public**(открытые или общедоступные)

**private**(закрытые или собственные)

**protected** (защищенные)

Если класс определялся с помощью ключевого слова **struct**, то по умолчанию все данные-членыи методы класса являются открытыми (**public**), то есть к любому элементу класса можно обратиться из внешней вызывающей программы или из другого класса.

В любом месте программы, где определен объект класса, можно получить доступ к его полям и методам с помощью уточненных имён, к полям класса anketa **-sotrudnik:: okl**

Если класс определяется с помощью ключа **class**, то все поля и методы класса являются закрытыми **(private)**, то есть недоступными для внешних обращений. Для обращения к закрытым данным из внешней (вызывающей) программы необходимо создавать «методы-посредники», которые обеспечат доступ к данным. Те поля данных и методы, которые необходимо открыть для внешнего доступа необходимо явно определить как **public**

**Друзья класса**

Механизм управления доступом с помощью ключа public не всегда удобен, ведь в этом случае public-переменные становятся общедоступны, то есть к ним можно обратиться из любой части программы. Механизм под названием «друзья класса» действует более гибко и позволяет предоставлять доступ избирательно. Например, можно предоставить доступ к закрытым переменным класса для отдельной функции или же для целого класса.

Друзья класса–это функции и классы, у которых есть полный доступ к классу, такой же, как у его собственных функций. Для получения прав друга, функция должна быть описана в классе со спецификацией friend.

Перепишем наши классы, используя ключевое слово class, для доступа к закрытым полям класса FIO, класс sotrudnik объявлен дружественным

class FIO

{

char \*familia, // указатели на строки с фамилией,

\*imya, // именем и отчеством

\*otchestvo;

friend class sotrudnik;

public:

//---------- конструкторы и деструкторы

FIO ();

~FIO ();

void define\_fio ();

}; //----------------- end FIO

class sotrudnik

{ FIO name; // ФИО сотрудника ( три указателя )

char \*dol; // должность ( указатель на строку)

float okl; // оклад (сумма оклада)

public:

//---------- конструкторы и деструкторы

sotrudnik();

~sotrudnik();

//----------методы

void define(FIO& fnam, char \*pd, float ok);

void display();

int get\_okl(){return okl; }// inline-функция

};//---------------- end sotrudnic

Для доступа к закрытым полям класса необходимы специальные методы, в нашем примере int get\_okl()

int \_tmain(int argc, \_TCHAR\* argv[])

{

sotrudnik a;

cout<<a.get\_okl()<<endl; // доступкполю okl обеспечиваетметод get\_okl()

**. . .**

}

**Статические поля класса.**

Статическое поле данных класса принадлежит всем объектам данного класса и существует в единственном экземпляре. При создании объектов класса статические поля не тиражируются, в отличие от остальных данных-членов класса. Следует отметить, что доступ к статическому полю возможен только после инициализации (выделения памяти и задания начального значения). Инициализировать статическую переменную нужно вне класса в глобальной области данных для того, чтобы экземпляры класса, создаваемые после этого имели к ней доступ.

Добавим в класс **sotrudnik** статическое закрытое поле **count** – общее количество созданных объектов и метод **get\_count()** – для внешнего чтения закрытого поля **count**

class sotrudnik

{ static int count;

FIO name; // ФИО сотрудника ( три указателя )

char \*dol; // должность ( указатель на строку)

float okl; // оклад (сумма оклада)

public:. . .

// доступп к закрытым полям

int get\_okl() {return okl;}

int get\_count(){return count;}

};//---------------- endsotrudnic

// конструкторпо-умолчанию

sotrudnik ::sotrudnik()

{name.familia=new char[60]; // выделение динамической памяти под строку

strcpy (name.familia," "); //инициализация

name.imya=new char[60];

strcpy (name.imya," ");

name.otchestvo=new char[60];

strcpy (name.otchestvo," ");

dol=new char[60];

strcpy (dol," ");

okl = 0;

count++; // счётчик объектов класса

 }

Ниже приведен весь текст рабочей программы, обратите внимание на то, что определения методов класса находятся вне класса, за исключением функций **get\_okl()** и **get\_count().** Если определение метода находится в классе, то метод является inline-функцией, то есть его код встраивается в программу при каждом вызове в процессе компиляции. У такой функции отсутствует стандартная процедура вызова.

#include"stdafx.h"

#include<iostream>

#include<string.h>

using namespace std;

class FIO

{ char \* familia, // указатели на строки с фамилией,

\*imya, // именем и отчеством

\*otchestvo;

friend class sotrudnik;

public:

// ИНТЕРФЕЙС КЛАССА FIO

// конструкторы

FIO (); //по-умолчанию

~FIO(); // деструктор

// ввод данных в FIO-объект с клавиатуры

void define\_fio ();

}; //-------------------- end FIO

class sotrudnik

{ FIO name; // ФИО сотрудника ( три указателя )

char \*dol; // должность ( указатель на строку)

float okl; // оклад (сумма оклада)

static int count; // число объектов класса sotrudnik

public:

// ИНТЕРФЕЙС КЛАССА sotrudnik

// конструкторы и деструкторы

sotrudnik();

~sotrudnik();

// методы

void define(FIO& fnam, char \*pd, float ok);

void display();

//доступ к закрытым полям класса

int get\_okl() {return okl;}

int get\_count(){return count;}

};//---------------- end sotrudnic

//===================== определения класса FIO

FIO::FIO ()

{cout<<"FIO::Конструктор по-умолчанию"<<endl;

familia=new char[60];

strcpy (familia," ");

imya=new char[60];

strcpy (imya," ");

otchestvo=new char[60];

strcpy (otchestvo," ");

}

FIO::~FIO()

{cout<<endl<<"FIO::Destructor!"<<endl;

delete[] familia;

delete[] imya;

delete[] otchestvo;

}

// ввод данных в FIO-объект с клавиатуры

void FIO::define\_fio ()

{char str[40];

cout<<endl<<"FIO\_object:";

cout<<"Фамилия :";

cin>>str;

familia=new char[strlen(str)+1];

strcpy (familia,str);

cout<<"Имя :";

cin>>str;

imya=new char[strlen(str)+1];

strcpy (imya,str);

cout<<"Отчество :";

cin>>str;

otchestvo=new char[strlen(str)+1];

strcpy (otchestvo,str);

}

//======================= определения класса sotrudnik

// конструктор по-умолчанию

sotrudnik::sotrudnik()

{cout<<"sotrudnik::Конструктор по-умолчанию"<<endl;

name.familia=new char[60]; // выделение динамической памяти под строку

strcpy (name.familia," "); // инициализация

name.imya=new char[60];

strcpy (name.imya," ");

name.otchestvo=new char[60];

strcpy (name.otchestvo," ");

dol=new char[60];

strcpy (dol," ");

okl = 0;

count++;// счётчик объектов класса

}

// деструктор

sotrudnik::~sotrudnik()

{cout<<endl<<"sotrudnik::Destructor!"<<endl;

delete[] name.familia;

delete[] name.imya;

delete[] name.otchestvo;

count--;// счётчик объектов класса

}

void sotrudnik::define(FIO& fnam, char \*pd, float ok)

{ name.familia = fnam.familia;

name.imya = fnam.imya;

name.otchestvo = fnam.otchestvo;

dol = pd;

okl = ok;

}

void sotrudnik::display()

{ cout<<endl;

cout<< name.familia<<" ";

cout<<name.imya;

cout<<" "<<name.otchestvo<<" , ";

cout<<dol<<" , ";

cout<<"оклад :"<<okl<<endl;

}

// область глобальных переменных

int sotrudnik::count=0; // создание статической переменной

Проведём несколько экспериментов с объектами класса **sotrudnik**

Пример 1: Работа со статическим объектом

int \_tmain(intargc, \_TCHAR\* argv[])

{// статические объекты chel, s1: память выделяется при трансляции

FIO chel;

sotrudnik s1;

chel.define\_fio();

s1.define(chel,"бухгалтер",12000);

s1.display();

cout<<"Число объектов sotrudnik::"<<s1.get\_count()<<endl;

system ("pause");

return 0;

}

Пример 2: Работа с динамическим объектом

int \_tmain(int argc, \_TCHAR\* argv[])

{FIO chel;

chel.define\_fio();

// динамический sotrudnik-объект: память выделяется во время выполнения

sotrudnik\* p; // указатель на тип sotrudnik

p= new sotrudnik;

p->define(chel,"бухгалтер",12000);

p->display();

cout<<"Число объектов sotrudnik::"<<s1.get\_count()<<endl;

system ("pause");

return 0;

}

Пример 3: Работа с массивами объектов

int \_tmain(int argc, \_TCHAR\* argv[])

{const int n=2;

FIO chel;

sotrudnik M[n];

char M\_dol[3][15]={"бухгалтер","слесарь","курьер"};

int M\_okl[3]={60000,15000,10000};

for (int i=0;i<n;i++)

{cout<<"сотрудник\_"<<i+1<<" :";

chel.define\_fio();

M[i].define(chel, M\_dol[i], M\_okl[i]);

}

for (int i=0; i<n; i++) M[i].display();

system ("pause");

return 0;

}

##### Контрольные вопросы

1. Что представляет собой класс?
2. Какие элементы класса Вы знаете?
3. Перечислите спецификаторы доступа, используемые в классе, для чего нужен каждый из них.
4. Как осуществляется доступ к элементам класса?
5. Для каких целей используется конструктор? Приведите пример.
6. Для каких целей используется деструктор? Приведите пример.
7. Что такое область видимости, применительно к данным-членам класса?
8. Как можно изменить область видимости с помощью ключевых слов?
9. Что такое друзья класса?
10. Сравните характер доступа, предоставленный с помощью ключа public, с доступом, предоставленным с помощью дружественных функций.
11. Статические поля класса, для чего они нужны, чем отличаются от прочих полей класса?

**Общие требования**

1. Для подготовки к лабораторной работе используйте данные из папок «Справочная информация»и «Теория»
2. Смотрите общие требования к лабораторной работе №1, а именно

* Общие требования к выполнению заданий
* Пример распечатки данных в табличном виде
* Пример диалогового интерфейса

1. Выполнить задание лабораторной работы №1 в стиле ООП. Для этого видоизменить программу, разработанную ранее (Задание 1).
2. Дополнить проект в соответствии с вашим вариантом задания (Задание 2).

**Требования к выполнению данной лабораторной работы**

1. В заданиях используются следующие сокращения:

pub– открытые данные (public),

pri – закрытые данные (private),

СД – статические данные (в единственном экземпляре, общие для всего класса)

НСД – нестатические данные тиражируются для каждого объекта

КК – конструктор копирования

ДФ – дружественная функция,

МК- метод класса

Например поле, помеченное СД, pri означает статическую переменную, размещенную в закрытой области.

1. Если поле никак не отмечено, то считать его открытым
2. Для текстовых данных в классе создавать указатели на строки, а в конструкторах динамически выделять память под строки, в деструкторах – возвращать в кучу.
3. В задании указаны методы, которые создаются обязательно, но вы не должны этим ограничиваться. Как правило, для работы требуются и другие методы, какие – решать вам.
4. Оценка лабораторной работы проходит по 2 критериям:

* оценка проекта (элементы класса: данные, методы и т.п.) - 50%
* оценка реализации (сама программа) – 50%

|  |  |
| --- | --- |
| **Номер компьютера** | **Варианты заданий** |
| 1, 11, 21 | **Пищевое производство (расчеты)**  **Задание 1**  Создать класс **food** со следующими элементами:  данные-члены класса:  char\* fam ; (pri) //Название изделия  int type; (pri) //тип изделия (1- булочка, 2-пирожок, 3-пирожное)  double weight; (pri) // вес  int quant; // количество  double cost; //стоимость  static int count; // счетчик существующих объектов класса  функции-члены класса   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск изделий по названию (МК) * Фильтр по типу (найти изделия заданного типа) (ДФ) * Сортировать по уменьшению стоимости (сначала более дорогие) (ДФ)   **Тестовая программа №1:**   * Массив объектов располагать в динамической памяти * При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) * Добавить несколько новых элементов (ввод с клавиатуры) * Выполнить все реализованные функции (поиск, фильтр, сортировка) * При выходе из программы запомнить измененные данные на диске |
|  | **Задание 2**   1. Создать новый класс **date,**   данные-члены класса:  int day; (pri) // день  int month; (pri)// месяц  int year; (pri)// год  функции-члены класса **date**: решайте сами, какие методы необходимы   1. Дополнить класс **food** встроенным объектом класса **date**  (pri)   ( дата производства)   1. Дополнить класс **food** следующими функциями-членами**:**  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструктор копирования  1. создать фильтры по дате изготовления (перегруженные методы класса):  * найти все изделия с заданной датой изготовления * найти все изделия с датой изготовления больше, чем заданная  1. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **food**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в массиве, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы |
| 2, 12, 22 | **Поезд (заказ билетов)**  **Задание 1**  Создать класс **ticket** со следующими элементами:  данные-члены класса:  char\* fio; // ФИО пассажира  char\* point\_from; (pri) // пункт отправления  char\* point\_to; (pri) // пункт прибытия  double price; // цена билета  static int count; // счетчик существующих объектов класса  функции-члены класса   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск билета по фио пассажира (МК) * Фильтр по цене (найти билеты с ценой более заданного значения) (ДФ) * Сортировать по алфавиту (поле point\_from) (ДФ)   **Тестовая программа №1:**   * Массив объектов располагать в динамической памяти * При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) * Добавить несколько новых элементов (ввод с клавиатуры) * Выполнить все реализованные функции (поиск, фильтр, сортировка) * При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **fio,**   данные-члены класса (все pri):  char \*familia, // указатели на строки с фамилией,  \*imya, // именем и отчеством  \*otchestvo;  функции-члены класса **fio**: решайте сами, какие методы необходимы   1. Изменить класс **ticket** : заменить char\* fio; // пассажир   на встроенный объект класса **fio**   1. Добавить в класса **ticket** следующие функции-члены**:**  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструктор копирования  1. Создать фильтр по **fio (**найти все билеты заданного пассажира**) (**ДФ) 2. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **ticket**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в массиве объектов, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы |

|  |  |
| --- | --- |
| 3, 13, 23 | **Почта (ценное письмо)**  **Задание 1**  Создать класс **letter** со следующими элементами:  данные-члены класса:  int ind1; // индекс получателя  char\* adr1; (pri) //адрес получателя  char\* name1; (pri) // Имя получателя  char\* adr2; (pri) // адрес отправителя  char\* name2; (pri) // Имя отправителя  double cost; (pri) // стоимость письма  static int count; // счетчик существующих объектов класса  функции-члены класса   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск письма по отправителю (МК) * Фильтр по индексу (ДФ) * Сортировать по увеличению стоимости (ДФ)   **Тестовая программа №1:**   * Массив объектов располагать в динамической памяти * При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) * Добавить несколько новых элементов (ввод с клавиатуры) * Выполнить все реализованные функции (поиск, фильтр, сортировка) * При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **man,** данные-члены класса :   char\* adr; (pri) //адрес  char\* name; (pri) // имя  функции-члены класса **man**: решайте сами, какие методы необходимы   1. Изменить класс **letter**: заменить отправителя и получателя на два объекта класса **man** 2. Добавить в класс **letter** следующие функции-члены**:**  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструкторы копирования  1. Создать фильтр по получателю **(**найти все письма заданного получателя**) (ДФ)** 2. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **letter**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в базе, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы |

|  |  |
| --- | --- |
| 4, 14, 24 | **База данных: Студент**  **Задание 1**  Создать класс **student** со следующими элементами:  данные-члены класса:  char\* fam; (pri) //Фамилия  char\* name; (pri) // Имя  char\* faculty // факультет  int grup // номер группы  static int count; // счетчик существующих объектов класса  функции-члены класса   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск студента по фамилии (МК) * Фильтр по номеру группы (найти студентов заданной группы) (ДФ) * Сортировать по алфавиту (ДФ)   **Тестовая программа №1:**   * Массив объектов располагать в динамической памяти * При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) * Добавить несколько новых элементов (ввод с клавиатуры) * Выполнить все реализованные функции (поиск, фильтр, сортировка) * При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **man,** данные-члены класса :   char\* fam; (pri) //Фамилия  char\* name; (pri) // Имя  функции-члены класса **man**: решайте сами, какие методы необходимы   1. Изменить класс **student,** два поля (фамилия и имя) заменить встроенным объектом класса **man** 2. Добавить в класс **student** следующие функции-члены**:**  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструкторы копирования  1. Создать фильтр по фамилии и имени (МК) 2. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **student**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в базе, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы |

|  |  |
| --- | --- |
| 5, 15, 25 | **Производство (поставщики)**  **Задание 1**  Создать класс **supplier** (поставщик) со следующими элементами:  данные-члены класса:  char\* firma; (pri) // название фирмы поставщика  char\* adr; (pri) // юридический адрес  int type; // 1- поставщик сырья, 2- поставщик оборудования  double balance; /\* сальдо (положительное - должны поставщику,  отрицательное-наоборот)\*/  static int count; // счетчик существующих объектов класса  функции-члены класса   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск поставщика по названию фирмы (ДФ) * Фильтр по типу (найти поставщиков заданного типа) (МК) * Сортировать по уменьшению сальдо (МК)   **Тестовая программа №1:**   * Массив объектов располагать в динамической памяти * При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) * Добавить несколько новых элементов (ввод с клавиатуры) * Выполнить все реализованные функции (поиск, фильтр, сортировка) * При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **date,** данные-члены класса:   int day; (pri) // день  char\* month; (pri)// месяц  int year; (pri) // год  функции-члены класса **date**: решайте сами, какие методы необходимы   1. Дополнить класс **supplier**: встроенным объектом класса **date** (дата поставки) 2. Добавить в класс **supplier** следующие функции-члены**:**  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструкторы копирования  1. создать фильтры по дате поставки (перегруженные методы класса):  * найти все поставки с заданной датой * найти все поставки с датой больше, чем заданная  1. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **supplier**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в базе, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы. |

|  |  |
| --- | --- |
| *6, 16, 26* | **Поликлиника**  **Задание 1**  Создать класс **clinic** со следующими элементами:  данные-члены класса:  char\* fio; //ФИО врача  int specialty; (pri) // специальность(1-терапевт, 2-кардиолог, 3- стоматолог)  int qualification; (pri) //квалификация (0-высшая, 1- первая, 2- вторая)  static int count; // счетчик существующих объектов класса  функции-члены класса   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск врача по фио (ДФ) * Фильтр по квалификации и специальности (найти врачей заданной специальности и квалификации) (МК) * Сортировать по алфавиту (МК)   **Тестовая программа №1:**   1. Массив объектов располагать в динамической памяти 2. При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) 3. Добавить несколько новых элементов (ввод с клавиатуры) 4. Выполнить все реализованные функции (поиск, фильтр, сортировка) 5. При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **fio,** данные-члены класса (все pri):   char \*familia, // указатели на строки с фамилией,  \*imya, // именем и отчеством  \*otchestvo;  функции-члены класса **fio**: решайте сами, какие методы необходимы   1. В классе **clinic** поле char\* fio; (//фио врача) заменить на встроенный объект класса **fio** 2. Добавить в класс **clinic** следующие функции-члены  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструкторы копирования  1. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **clinic**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в базе, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы. |

|  |  |
| --- | --- |
| 7, 17, 27 | **Склад (товары)**  **Задание 1**  Создать класс **storage** со следующими элементами:  данные-члены класса:  char\* name ; (pri ) //Название товара  double price; (pri ) // цена  int quant; // количество  int num; // номер секции  static int count; // счетчик существующих объектов класса  функции-члены класса   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск товара по названию (МК) * Фильтр по номеру секции (найти изделия заданной секции) (МК) * Сортировать по уменьшению количества (ДФ)   **Тестовая программа №1:**   1. Массив объектов располагать в динамической памяти 2. При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) 3. Добавить несколько новых элементов (ввод с клавиатуры) 4. Выполнить все реализованные функции (поиск, фильтр, сортировка) 5. При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **date,** данные-члены класса:   int day; (pri) // день  int month; (pri)// месяц  int year; (pri)// год  функции-члены класса: решайте сами, какие методы необходимы   1. Дополнить класс **storage** встроенным объектом класса **date** 2. Добавить в класс **storage** следующие функции-члены  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструкторы копирования  1. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **storage**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в базе, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы. |

|  |  |
| --- | --- |
| 8, 18, 28 | **Банк (депозит)**  **Задание 1**  Создать класс **bank** со следующими элементами:  данные-члены класса:  char\* fam; (pri) //Клиент: Фамилия  char\* name; (pri) // Имя  int account; //Номер счета  double sum; // Сумма на счете (сумма депозита )  static int count; // счетчик существующих объектов класса  функции-члены класса:   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск клиента по фамилии (ДФ) * Фильтр по сумме на счете (найти клиентов с суммой больше заданной) (МК) * Сортировать по алфавиту (ДФ)   **Тестовая программа №1:**   1. Массив объектов располагать в динамической памяти 2. При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) 3. Добавить несколько новых элементов (ввод с клавиатуры) 4. Выполнить все реализованные функции (поиск, фильтр, сортировка) 5. При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **date,** данные-члены класса:   int day; (pri) // день  char\* month; (pri)// месяц  int year; (pri) // год  функции-члены класса: решайте сами, какие методы необходимы   1. Дополнить класс **bank** встроенным объектом класса **date**   (дата открытия счета)   1. Добавить в класс **bank** следующие функции-члены**:**  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструкторы копирования  1. создать фильтры по дате открытия счета (перегруженные методы класса):  * найти все депозиты с заданной датой (МК) * найти все депозиты с датой больше, чем заданная (ДФ)  1. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **bank**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в базе, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы. |

|  |  |
| --- | --- |
| 9, 19, 29 | **Работник (зарплата)**  **Задание 1**  Создать класс **worker** со следующими элементами:  данные-члены класса:  char\* fam; (pri) // Работник: Фамилия  char\* name; (pri) // Имя  int department; // отдел  double salary; // оклад  static int count; // счетчик существующих объектов класса  функции-члены класса:   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск работника по фамилии (МК) * Фильтр по отделу (найти работников заданного отдела)(ДФ) * Сортировать по возрастанию оклада (МК)   **Тестовая программа №1:**   1. Массив объектов располагать в динамической памяти 2. При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) 3. Добавить несколько новых элементов (ввод с клавиатуры) 4. Выполнить все реализованные функции (поиск, фильтр, сортировка) 5. При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **fio,** данные-члены класса (все pri):   char \*familia, // указатели на строки с фамилией,  \*imya, // именем и отчеством  \*otchestvo;  функции-члены класса **fio**: решайте сами, какие методы необходимы   1. В классе **worker** два поля: char\* fam; // Фамилия   char\* name; // Имя  Заменить на встроенный объект класса **fio**   1. Добавить в класс **worker** следующие функции-члены**:**  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструкторы копирования  1. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **worker**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в базе, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы. |

|  |  |
| --- | --- |
| 10, 20, 30 | **Преподаватель**  **Задание 1**  Создать класс **prepod** со следующими элементами:  данные-члены класса:  int department; // кафедра (1-ИПОВС, 2-ВМ, 3-ВТ)  char\* fio; //ФИО преподавателя  int status; //ученое звание ( 1- преподаватель, 2- доцент,3-доктор наук)  static int count; // счетчик существующих объектов класса  функции-члены класса:   1. Создать обязательные функции, указанные в общих требованиях (МК). 2. Создать функции для данного варианта:  * Поиск преподавателя по фио (МК) * Фильтр по статусу (найти преподавателей заданного статуса) (ДФ) * Сортировать по алфавиту (ДФ)   **Тестовая программа №1:**   1. Массив объектов располагать в динамической памяти 2. При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) 3. Добавить несколько новых элементов (ввод с клавиатуры) 4. Выполнить все реализованные функции (поиск, фильтр, сортировка) 5. При выходе из программы запомнить измененные данные на диске |
| **Задание 2**   1. Создать новый класс **fio,** данные-члены класса (все pri):   char \*familia, // указатели на строки с фамилией,  \*imya, // именем и отчеством  \*otchestvo;  функции-члены класса **fio**: решайте сами, какие методы необходимы   1. В классе **prepod** поле char\* fio; заменить на встроенный объект класса **fio** 2. Добавить в класс **prepod** следующие функции-члены**:**  * перегруженные конструкторы (по умолчанию, с параметрами по умолчанию) * конструкторы копирования  1. Изменить все МК и ДФ в соответствии с дополненными элементами   **Тестовая программа №2:**   * Выполнить все действия **Тестовой программы №1** * Создать объект класса **prepod**, используя конструктор с параметрами – это эталонный объект (ЭО) * Найти ЭО в базе, если не найдено, то дополнить массив на основе копирования ЭО. Если найдено, то распечатать найденные элементы. |